Teacher Timetable Extraction & Visualization Platform

ARCHITECTURAL DESIGN PLAN

Technical Architect Assessment — Learning Yogi

Presented By

LALATENDU MISHRA

ENTERPRISE ARCHITECT

1. Introduction

Teachers frequently store their class timetables in diverse formats: scanned images, PDFs, Word documents, photographs, and stylized tables. The goal of this system is to allow teachers to upload any timetable document and transform it into a clean, structured timetable representation within our frontend UI.

This document outlines the **end-to-end architecture**, **recommended technologies**, **LLM integration strategy**, **data schema**, **error handling**, and **future-proofing approach** for the system.

1.1 Assumptions

To ensure clarity and establish realistic boundaries for the architectural design, the following assumptions have been made regarding system behavior, user patterns, input characteristics, and operational constraints:

1. Input Document Assumptions

- Teachers primarily upload **timetables in English**.
- Supported formats include **PDF, JPEG, PNG, and DOCX**.
- Timetables are typically **1–3 pages**, with most being a **single page**.
- Uploaded files are assumed to be **under 10–15 MB** in size.
- The majority of documents follow a **grid/table-like structure**, even if scanned or stylized.

2. User Behavior Assumptions

- Teachers upload timetables **infrequently**, typically at the start of a term or academic session.
- Uploads occur sporadically; system load is not continuous but may spike during onboarding periods.
- Teachers may review or edit extracted timetables after initial processing (future enhancement).

3. System Processing Assumptions

- Vision LLM models will be available with reasonable latency (< 6–8 seconds per request).
- OCR engines are accessible with stable performance, assuming sufficient CPU.

- Internet connectivity between the backend server and the LLM API provider is reliable.

4. Output & Schema Assumptions

- All extracted timeblocks adhere to the defined **JSON schema**.
- Weekdays follow a standard set (Monday–Friday), unless explicitly detected otherwise.
- Time formats can be normalized to **HH:MM (24-hour format)**.
- Subjects will be extracted as free text and not validated against a predefined list.

5. Deployment & Environment Assumptions

- Stateless backend services allow for horizontal scaling.
- Storage for both raw uploads and processed JSON is available and scalable (e.g., S3 + Postgres).
- Logging and observability tools (CloudWatch, Grafana, ELK, etc.) are available.

6. Ethical & Regulatory Assumptions

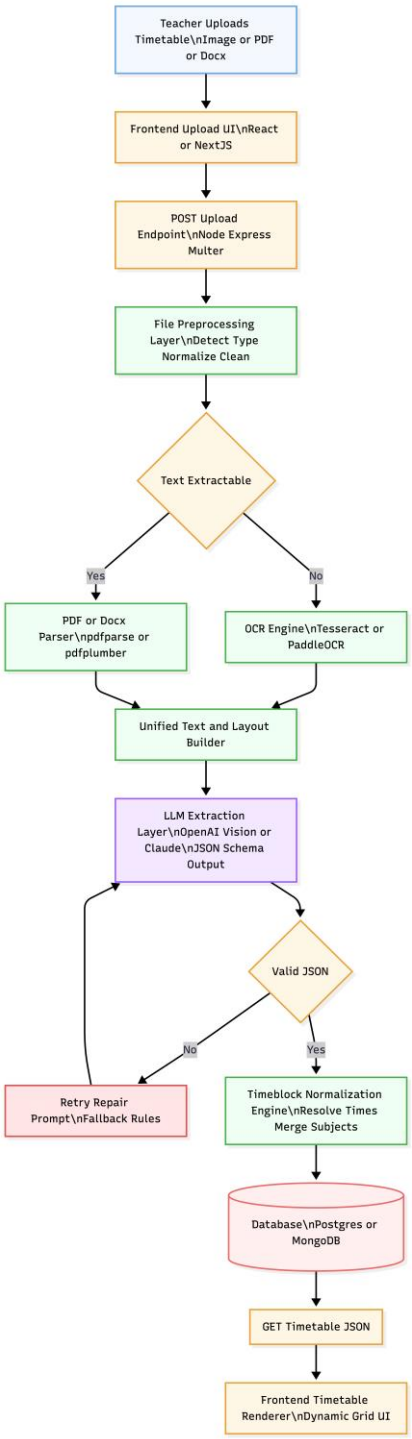
- Teachers consent to the processing of uploaded documents.
- Timetables do not contain sensitive student data or personal identifiers beyond what is necessary.
- All AI interactions adhere to provider data usage policies (e.g., no data used for model training).

2. End-to-End Workflow Overview

High-Level Pipeline

1. **File Upload (Frontend)**
2. **Backend Ingestion & Preprocessing**
3. **OCR / Native Text Extraction**
4. **LLM-driven Timetable Interpretation & Structuring**
5. **Timeblock Normalization & Validation**
6. **Storage in Database**
7. **Frontend Rendering of Standardized Timetable**

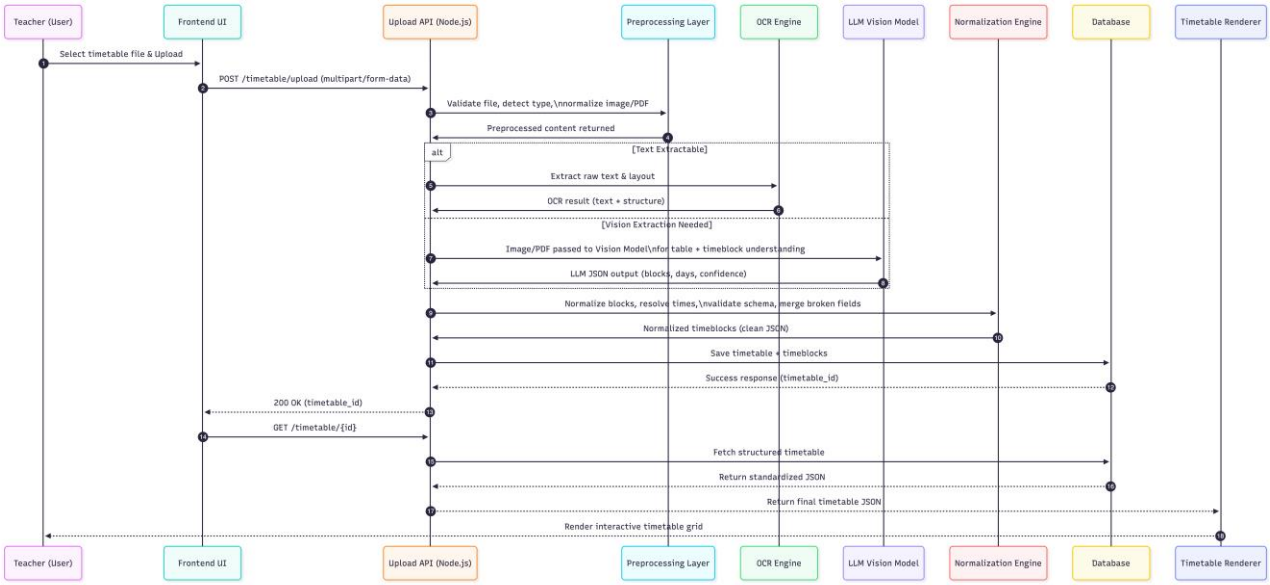
3. Annotated Architecture Diagram



3.1 Sequence Diagram — End-to-End System Interaction

While the architecture diagram provides a high-level view of the system components, the following sequence diagram illustrates the runtime interaction between the teacher's upload action, backend processing pipeline, OCR/LLM subsystems, and final data delivery to the frontend UI.

This expands on the flow by showing synchronous vs. asynchronous execution, error recovery points, and the order of operations within the extraction pipeline.



4. Recommended Tools, Frameworks, and Programming Languages

4.1 File Ingestion & Preprocessing

Component	Recommended Tool	Reason
Backend server	**Node.js + Express**	Fast to implement, supports file uploads easily
File upload	**Multer**	Battle-tested form-data/file ingestion
PDF parsing	**pdf-parse / pdfplumber**	Extracts structured text/layout from PDFs
DOCX parsing	**docx / mammoth**	Reliable DOCX text/table extraction
Image preprocessing	**Sharp**	Resize, normalize brightness, denoise before OCR

4.2 OCR & Document Parsing (Tables, Images)

Input Type	Tool	Benefit
Scanned JPG/PNG	**Tesseract OCR / PaddleOCR**	Robust open-source OCR
Mixed layout PDFs	**OpenAI Vision / Claude Vision**	Extracts both text and structure reliably
Heavily styled tables	**LLM Vision**	Detects grid relationships better than pure OCR

4.3 LLM Integration

OpenAI GPT-4o Vision or **Claude 3 Sonnet Vision**

For:

- Table structure understanding
- Timeblock classification
- Extracting start/end times
- Interpreting multi-line cells
- JSON generation

Why LLM Vision?

Timetables vary widely (scanned, stylized, coloured). Vision models eliminate the need for complex bespoke OCR pipelines.

4.4 Backend Orchestration

Node.js with:

- Express
- Axios/Fetch for LLM calls
- Zod/JSON schema validation
- Winston for logs
- Redis (optional) for caching repeated OCR calls

4.5 Frontend Rendering

Recommended stack:

- **React** (or Next.js)

- **Tailwind CSS** for flexible grid layouts
- **CSS Grid** to dynamically construct time-based columns/rows
- **Recharts / D3** (optional) for timeblock visualization
- State management: **Zustand** or **Redux Toolkit**

5. Suggested Database Schema

5.1 Entities

- **Teacher**
- **Timetable** (one per upload)
- **Timeblock** (child of Timetable)

5.2 Schema (SQL Model)

```
TABLE teachers (
  id UUID PRIMARY KEY,
  name TEXT,
  email TEXT,
  created_at TIMESTAMP
);

TABLE timetables (
  id UUID PRIMARY KEY,
  teacher_id UUID REFERENCES teachers(id),
  original_filename TEXT,
  uploaded_at TIMESTAMP,
  status TEXT,
  notes TEXT
);

TABLE timeblocks (
  id UUID PRIMARY KEY,
  timetable_id UUID REFERENCES timetables(id),
  day_of_week TEXT,
  start_time TIME,
  end_time TIME,
  subject TEXT,
  notes TEXT,
  raw_text TEXT,
```

```
  confidence_score FLOAT
);
```

Highlights:

- raw_text stores extracted cell text → auditability
- confidence_score helps UI highlight uncertain entries
- Decoupled structure → reusable for multiple upload attempts

6. LLM Integration Strategy

6.1 Where LLM Is Used

LLM is used in **three critical stages**:

1. **Vision Understanding**
    a. Interpret grids, rows, merged cells
    b. Extract start/end times
    c. Recognize subjects, notes, annotations
2. **JSON Schema Structuring**
    a. Ensure consistent output format
3. **Error Repair / Retry Logic**
    a. Fix malformed JSON
    b. Clarify ambiguous time formats

6.2 Prompt Strategy

Prompt 1: Vision Understanding Prompt

- Pass entire image/PDF page
- Ask for:
    - Timeblocks
    - Day grouping
    - Start/end times
    - Subject text
    - Confidence score

Prompt 2: JSON Schema Enforcement

Provide a schema:

```
{
 "type": "array",
 "items": {
  "day": "string",
  "blocks": [
   {
    "start": "HH:MM",
    "end": "HH:MM",
    "subject": "string",
    "notes": "string"
   }
  ]
 }
}
```

Prompt 3: Repair Prompt

If JSON fails validation:

- Ask LLM to regenerate only the invalid field
- Provide validator error message for correction

6.3 Ensuring Accuracy & Reproducibility

- **Few-shot examples** for teacher-specific timetable formats
- **Structured JSON schema** to constrain output
- **Retry mechanism** (max 3 attempts)
- **Cross-check with rule-based heuristics**:
    - Times must be chronological
    - Subjects cannot overlap
    - Empty rows ignored

7. Error Handling & Fallback Strategy

7.1 Bad Uploads

- Empty file → return HTTP 400
- Unsupported format (EPS, TIFF, etc.) → 415 Unsupported Media
- Image resolution too low → attempt auto enhancement, else error

7.2 Ambiguous Data

- Missing end time → infer from next block
- Missing subject → label as "Unknown" but log in notes
- Overlapping times → highlight as low confidence

7.3 LLM or OCR Failure

Fallback order:

1. OCR only
2. LLM vision only
3. OCR + LLM combined
4. Minimal extraction (text only + heuristic splitting)
5. Return partial timetable with warnings

8. System Flexibility & Futureproofing

Built-in extensibility:

- Modular pipeline:
  Preprocessing → OCR → LLM → Normalization → Storage
- Can replace OCR library without changing the rest
- Can switch to different LLM providers (Claude / Gemini)
- Schema versioning for future keys (e.g., room number, teacher name)
- Multi-page timetables supported
- Can easily add:
  - Timetable editing UI
  - Auto-detection of school term periods
  - Student view

- Parent notifications

8.1 Tradeoff Analysis & Alternatives Considered

The following evaluates key architectural decisions and the alternatives considered.

Node.js vs. Python for Backend Orchestration

Node.js Selected Because:

- Native non-blocking I/O suits file uploads and async LLM/OCR calls.
- Faster to scaffold APIs, ideal for rapid iteration cycles.
- Rich ecosystem for web applications and document processing.
- Lower cold-start times in serverless deployments.

Python Considered But Not Chosen Because:

- Strong for ML and PDF parsing, but less performant in high-concurrency web APIs.
- Would require maintaining separate services (OCR in Python, API in Node), increasing operational complexity.

Conclusion: Node.js provides a pragmatic balance between speed, concurrency handling, developer productivity, and integration simplicity.

OpenAI/Claude Vision vs. Pure OCR

Vision LLMs Selected Because:

- Handle stylized, rotated, handwritten, and irregularly spaced timetables.
- Understand *semantic structure* (e.g., merged cells, multi-line subjects).
- Produce structured JSON directly, reducing downstream processing.

Pure OCR Considered But Not Chosen Because:

- OCR-only approaches struggle with table borders, merged cells, and multi-row spans.
- Heavily dependent on document quality, lighting, resolution.
- Require complex heuristics to reconstruct layout.

Conclusion: Vision LLMs dramatically reduce complexity, improve accuracy, and provide future adaptability.

JSON Schema vs. XML or Protobuf

JSON Schema Selected Because:

- Easy to validate, easy to debug, human-readable.
- Works naturally with JavaScript/Node backend and React frontend.
- Ideal for flexible, evolving structures such as timetables.

XML Considered But Not Chosen Because:

- Verbose and harder to work with for small payloads.
- Slower parsing and less intuitive for modern frontend stacks.

Proto buf Considered But Not Chosen Because:

- Highly efficient but not necessary for small document-oriented payloads.
- Adds serialization overhead and reduces readability during debugging.

Conclusion: JSON Schema strikes the right balance of clarity, flexibility, and robustness.

SQL vs. NoSQL Database

Relational Database Selected Because (SQL):

- Timeblocks are highly structured and relational by nature.
- Easier to enforce constraints, uniqueness, and consistency.
- Supports filtering by teacher, day, subject, or historical version.

NoSQL Considered Because:

- Flexible schema for evolving formats.
- Faster for document-style storage.

8.2 Complexity Analysis & Throughput Expectations

Average Extraction Pipeline Time

- PDF/DOCX with clean structure: **3–5 seconds**
- Scanned images with OCR: **5–10 seconds**
- Vision LLM-only parsing: **5–8 seconds**

Expected Concurrency Levels

- Typical usage patterns by teachers imply:
    - **10–50 concurrent uploads** during peak times
    - Horizontal scaling recommended via stateless API nodes

File Size Handling

- Recommended upload limit: **10–15 MB** per file
- Above this threshold, async processing or pre-compression advised

Token Usage & Cost Estimates

- Vision LLM call: **500–2,000 tokens** depending on timetable density
- JSON validation prompt: **50–150 tokens**
- Repair attempts: **150–400 tokens**

Pipeline Complexity

Overall pipeline complexity is **O(n)** relative to the number of text elements extracted.
Each stage is independent, ensuring predictable scaling behaviour.

8.3 SLA / SLO Definitions

To ensure production readiness, clear service targets are defined.

Service Level Objectives (SLOs)

- **95%** of timetables extracted within **10 seconds**
- **99%** Vision/OCR extraction accuracy for clean documents
- **99%** schema validation success after LLM correction loop

Service Level Agreements (SLAs)

- **99.9% uptime** for Upload API
- **99.5% availability** for LLM integration endpoints
- **Maximum timeout** for LLM requests: **15 seconds**

Error Budget

- Up to **1% monthly extraction failures** acceptable before triggering corrective action
- Retries must not exceed 2 attempts per extraction

Retry & Timeout Policies

- OCR timeout: **5 seconds**, retry once
- LLM timeout: **15 seconds**, retry twice
- Network retry: exponential backoff, max 3 attempts

8.4 Regulatory & Ethical AI Considerations

Given that teacher timetables may contain personal or location-based information, responsible AI practices must be applied.

Data Minimization

- Only extract fields directly relevant to timetable structure.
- Personal identifiers are removed or excluded from logs.

No Training on User Data

- Uploaded timetables **must not** be used for LLM training or improvement.
- Strict compliance with provider data-use guidelines (OpenAI/Anthropic).

Privacy Compliance

- Align with GDPR principles:
    - Right to access
    - Right to deletion
    - Minimal retention of uploaded documents
- Raw uploads stored temporarily, then purged after 30 days.

Explainability & Transparency

- Confidence scores provided for each extracted block.
- Ambiguous or low-confidence areas clearly highlighted for teacher review.

Secure Processing

- All documents scanned for malicious content before processing.
- TLS encryption for upload and retrieval endpoints.

9. Conclusion

This architectural design ensures:

- High robustness across arbitrary timetable formats
- Clean separation of concerns
- LLM Vision used strategically, not excessively
- Validated, reliable, reproducible JSON extraction
- A scalable and future-proof backend
- A flexible, intuitive frontend UI for teachers

This approach balances **AI-driven intelligence** with **deterministic engineering**—ensuring both accuracy today and extensibility for tomorrow's needs.

SECTION 2

10. Non-Functional Requirements (NFRs)

A robust timetable extraction system must uphold several critical non-functional requirements to ensure consistent, scalable, and predictable performance across diverse real-world timetable formats.

10.1 Performance

- The system should process standard PDF/DOCX timetables within **3–5 seconds**.
- Scanned images requiring OCR may take up to **8–10 seconds**, with asynchronous processing recommended for heavier inputs.
- LLM calls must be optimized using structured prompts and schema-restricted outputs to minimize token usage and latency.

10.2 Scalability

- Backend services must remain **stateless**, enabling horizontal scaling behind a load balancer.
- File storage and processing (OCR, LLM) should be decoupled, allowing independent scaling of compute-heavy components.
- Use object storage (e.g., AWS S3) for uploaded files to avoid local disk dependency.

10.3 Security

- Strict validation of MIME types and file extensions.
- Antivirus scanning for uploaded documents before processing.
- Ensure no sensitive teacher data is logged or exposed.
- Enforce authentication and rate limiting on API endpoints.

10.4 Reliability & Resilience

- Automatic retries for transient LLM or OCR failures.
- Fallback strategies (OCR-only, text-only heuristics) to ensure partial results are still produced.
- Dead-letter queue for files that fail processing after multiple attempts.

10.5 Cost Efficiency

- Use LLM Vision only when needed; default to OCR + lightweight model for simple documents.
- Introduce caching for identical or repeated uploads.
- Switch between "fast" and "accurate" model modes based on system load or user preference.

11. Observability & Monitoring Strategy

Observability is essential for ensuring reliable extraction quality, debugging failures, and identifying patterns in problematic uploads.

11.1 Metrics

Capture the following metrics for each upload:

- Total extraction time
- OCR duration
- LLM duration
- Schema validation success/failure
- Timeblock completeness score
- Number of ambiguous fields requiring fallback
- Confidence score distribution across timeblocks

11.2 Logs

Adopt a structured logging strategy:

- **INFO**: upload received, extraction pipeline started, pipeline completed
- **WARN**: missing fields, low-confidence OCR, partial extraction
- **ERROR**: OCR failure, LLM unresponsive, schema violations

11.3 Dashboards

Implement dashboards for:

- Processing throughput
- Failure rates per document type
- Average extraction time by source format
- LLM token usage & cost trends

11.4 Alerts

Trigger alerts when:

- LLM failure rate exceeds threshold
- OCR accuracy drops below expected baseline
- Extraction time spikes beyond SLA

12. AI Evaluation & Quality Framework

To maintain consistent quality across varied timetable formats:

12.1 Accuracy Metrics

- **Block detection accuracy:** % of correctly identified sessions

- **Time precision:** accuracy of extracted start/end times
- **Subject text accuracy:** closeness of extracted text to source
- **Layout interpretation accuracy:** row/column detection correctness

12.2 Feedback Loop

Enable teachers to:

- Correct extracted timeblocks
- Add missing sessions
- Merge/split incorrectly detected blocks

Store corrections to:

- Improve prompts
- Build a fine-tuning dataset for future custom model training
- Identify recurring extraction issues

12.3 Model Selection Strategy

Use a tiered approach:

- **Tier 1:** Fast Vision model for extraction
- **Tier 2:** More accurate Vision model for ambiguous cases
- **Tier 3:** Human-in-the-loop or manual correction (future feature)

13. Architectural Principles

This solution is guided by the following architectural principles:

1. **Modularity** — Each pipeline stage (OCR, LLM, normalization) functions independently.
2. **Fail-Safe Defaults** — Always return partial extraction rather than failing fully.
3. **LLM as an Assistive Layer** — Use deterministic logic where possible; use LLMs only where structure is unclear.
4. **Schema-First Design** — Enforce a strict JSON schema to ensure consistent output.
5. **Separation of Concerns** — Frontend rendering completely decoupled from backend extraction logic.
6. **Forward Compatibility** — Designed to support additional formats: spreadsheets, school-specific templates, handwritten notes.

7. **Explainability** — Each extracted block includes confidence scores and raw text for auditing.

14. Technical Risks & Mitigation Strategies

Risk 1: LLM hallucinations or inconsistent interpretation

Mitigation:

- Enforce JSON schema
- Provide strong few-shot examples
- Use correction prompts and validation loops

Risk 2: Poor OCR quality for low-resolution scans

Mitigation:

- Preprocess images (denoise, deskew)
- Switch to Vision LLM mode when OCR confidence drops
- Highlight uncertain fields for teacher review

Risk 3: Inconsistent timetable layouts

Mitigation:

- Use Vision models for grid interpretation
- Add heuristics for row/column alignment
- Incrementally build layout-agnostic extraction logic

Risk 4: Large or corrupted files

Mitigation:

- Enforce upload limits
- Async processing with progress updates
- Graceful fallback with user notifications

15. Data Lifecycle & Governance

15.1 Storage Architecture

- Raw uploaded files → Object storage (S3)
- Extracted structured data → Database
- Logs/metrics → Monitoring system

15.2 Retention Policies

- Raw files: retain for 30 days
- Extracted metadata: retain indefinitely for reuse
- Audit logs: retain for 90–180 days

15.3 Versioning

- Each upload creates a *new version* of a teacher's timetable
- Versioning allows rollback and historical comparisons

15.4 Data Privacy

- Strip teacher names and sensitive metadata where unnecessary
- Comply with global data privacy norms (GDPR-like principles)

16. API Contract & Schema Definitions

16.1 Upload API

POST /api/timetable/upload
Content-Type: multipart/form-data
Body: file

16.2 Response Schema

```
{
 "timetable_id": "uuid",
 "days": [
  {
```

```
    "day": "Monday",
    "blocks": [
     {
       "start": "08:55",
       "end": "10:10",
       "subject": "Maths",
       "notes": null,
       "confidence": 0.94
     }
    ]
   }
 ]
}
```


17. Deployment Architecture Overview

Recommended Stack

- **Dockerized services** for backend + OCR engine
- **CI/CD** pipeline for automated build/test/deploy
- **Environment separation:** dev, staging, production
- **Feature flags:** enable/disable LLM, OCR modes

Infra Considerations

- Horizontal scaling for Node backend
- Dedicated worker nodes for OCR/LLM-heavy tasks
- Load balancing with Nginx or cloud LB

18. Future Enhancements Roadmap

Short-Term

- Teacher UI for easy correction of extracted timetable
- Specialized visual editor for timeblocks
- Confidence-based highlighting

Mid-Term

- School-wide timetable dashboard
- Automatic clash detection
- AI suggestions for optimization (teacher load balancing)

Long-Term

- Fine-tuned custom Vision model for timetables
- Integration with school ERP systems
- Multi-language timetable understanding

19. UX Considerations

- Display low-confidence items in yellow for teacher verification
- Drag-and-drop editing of timeblocks
- Smooth visualization even for highly irregular time ranges
- Mobile-friendly timetable previews for teachers on the go